VOICE ACCESS TO ELECTRONIC MAIL

by

Caren Hope Baker

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1983

Signature of Author _____
Department of Electrical Engineering and Computer Science
May 20, 1983

Certified by _____
Patrick Purcell
Thesis Supervisor

Accepted by _____
David Adler
Chairman, Departmental Committee on Theses

Voice Access To Electronic Mail

by

Caren Hope Baker

Submitted to the Department of Electrical Engineering and Computer Science on May 20, 1983 in partial fulfillment of the requirements for the degree of Bachelor of Science.

Abstract

Continued growth of electronic mail could be ensured by providing verbal access to the mail. This thesis describes the software which comprises the user-mail interface. Previously, only pre-recorded messages could be accessed verbally. This interface is unique in that it provides access to written mail. This is difficult since conventional message format is not good for vocal transmission. The software obtains and rearranges the message text to facilitate comprehension as a vocal message. At the user's request, the synthesizer will read a selected message, repeat it, or reply with set answers. By providing these options, the verbal mail system gives the listener the same flexibility as a reader in the use of electronic mail.

Thesis Supervisor: Patrick Purcell
Title: Visiting Associate Professor of Computer Graphics

Acknowledgements

I would like to thank a number of people for their patience and guidance:

Chris Schmandt, who gave me many ideas which are embedded in this project, including the topic itself.

Patrick Purcell, my thesis supervisor, for his help and support through out the term.

Barry Arons, who provided many helpful suggestions incorporated into this work.

Leo Hourvitz, who wrote the interface into the mail system.

Mark Vershel, who helped me find a thesis.

Julie Foster, Frank DiTaranto, Melissa Miller, and Brenda Reale, for helping to edit this thesis.

Table of Contents

Chapter 1
Introduction

Current electronic mail systems, such as the one at MIT's Architecture Machine Group, allow users access only through computer terminals. A person's messages, stored in memo format, are kept in his "mailbox" and can be read at any time from a terminal. But the rapid spread of electronic mail systems [4] requires easier ways to get one's messages, for example, via push-button telephones. To do this for MIT Architecture Machine Group's system requires major hardware and software modifications.

At present, a typical message on a terminal may read:

> #1
> Date: Friday 20 May 1983 02:59:15 EDT (NM + 4d 9h 16m 15s)
> From: Chris Schmandt <geek@mit-pamela>
> Sender: Mark A. Vershel <mav@sri-kl>
> Subject: Mail system
> To: caren
> cc: geek
>
> When using a terminal, a person can read messages and delete them afterward or keep them for future reference. The user can read messages in any order or select an option that summarizes the mail. The summary states the message number, the sender, the date, and the subject. If the sender has omitted the subject from the header, the summary contains the first 40 characters of the message.
>
> But hearing voice and reading text differ in several ways. For one, a spoken message takes longer to understand than a written one; you can not go back and reread. Furthermore, you must listen to a message all the way through; you can not scan the

> whole text or skip over unimportant parts. The mail system's structure must change before voice access will work.

My thesis project is part of an analog-digital telecommunications system that will allow users to hear telephone messages or electronic mail messages over the telephone. A voice recognizer and a text-to-speech voice synthesizer will communicate with the caller. To login, users will call the computer and enter their passwords. They can then "listen" to mail or phone messages. I was responsible for providing the software interface for the verbal mail system.

Previous work has been done allowing users to receive and send voice mail using a phone [8,9]. However, these systems use stored voice messages instead of text. My system is unique in that it converts text to speech, thus increasing access to the existing mail system.

Chapter 2
Motivation

Traditionally, business communications have been by the following methods: face to face, the mails, and the telephone. Disadvantages with these methods include problems with location, wasted time, lack of records, and unnecessary formalities [4]. Electronic mail is becoming more popular because it minimizes these difficulties.

Location and time are major disadvantages of the telephone and the face to face method. It takes time to locate people. It is often the case that when you call someone they are not by their phone. Then if you leave a message, and the person returns the call, you might be out. This process can continue for a long time. After reaching the person, the original call may have lost its importance. Similarly, written mail is inappropriate for urgent messages because of the time required for delivery.

Timing is difficult with traditional methods. You can contact people only at certain times. People are not always willing to be disturbed. You can call or see people only if they are free. The time zone differences also limit the time when you can reach someone. If you are calling someone in a different time zone, it may be hard to find a time when both people are in.

Automatic record keeping is not part of communication by traditional methods, which is important in the business world. There is no record after a telephone call or a meeting. In order to keep track of what was said, you have to write everything down. Mail or memos can be used as records, but

they are difficult to arrange because of their inconsistaent format.

It is difficult to broadcast messages to everyone, as each person has to be contacted individually. This redundancy can be avoided.

Long-windedness is another problem with all three means. For example, when you contact a person, you cannot just say the message, you tend to talk about general things first. Also, mail has to be written in a business letter format, which is time consuming and may require secretarial help.

Electronic mail avoids many of these disadvantages. It is informal, thus you can get right to the point. It is easy to use because it has an on-line editor and can be used at any time. There is no problem with location or time. Once the recipient returns to their terminal they will get the message. It is also fast; the message is immediately sent to the person. A record is automatically made of the message, since it is stored in the user's mail box and can be logged. It is easy to broadcast messages since you can send one message to many people.

Unfortunately, electronic mail cannot always be accessed because of the need for a terminal or modem. This creates the need for alternative methods. Voice access using the existing telephone system will increase the use of electronic mail.

Chapter 3
Design Considerations

The differences between hearing voice and reading text became apparent during the design process. First a user can reread portions of written text that are not completely understood, but cannot hear verbal messages again. Also, a user can quickly scan written text to pick out important details, but must listen to the entire spoken text to hear all important points. In this way, verbal mail can take longer to access than written mail. The verbal mail interface must be adapted to take into account these differences.

In using verbal mail the ability to reread ambiguous phrases is lost. To compensate for this, I have created a command that repeats the last phrase spoken. The first time it is repeated, it is spoken slowly, while the second time, each word is spelled out. This helps the user understand the voice synthesized words.

To alleviate the scanning problem, the verbal mail messages are grouped by sender, starting with the sender who has sent the most messages. This order is more efficient for two reasons. One, the user should usually know what each person's messages are about and can decide if he wants to hear them. Also, the user can reply to all of the message from one sender at one time. This is especially useful when more than one message relates to the same topic.

The first implementation tried to imitate written electronic mail using the new groupings, but ended up being too wordy. First the user heard a list

containing the names of the senders and number of messages they sent. Then for each individual sender, the system provided a summary of his messages. Then the user was provided with the message text. In this way the design was similar to the written electronic mail system; however, the large amount of information given confused the user.

The latest version eliminates the full summaries by starting the message with its topic. This also helps to alleviate the scanning problem because the user can hear the topic and decide whether or not to continue listening to the message. In addition, the user can easily recognize the message by its topic.

Another problem with the first version is that it asked questions at every decision point. Redundant questions in written mail can be skipped over by the eye, whereas all parts of verbal mail must be heard. This process is very slow. In the second version each question was eliminated and replaced by a three-second pause, that informed the user that a decision could be made then. Unfortunately a three-second pause at each decision point wasted time. Finally the pauses were reduced to one second. A one-second pause is just long enough for the user to give a command, but minimizes wasted time.

Because the pause is so short, there is a chance that a spoken command will not be entered during the pause. Also, the voice recognizer does not process the command immediately, so that further delay is incurred. A time margin must be allowed for each command so that a command entered late will be executed correctly. This is accomplished by allowing commands to be entered while the synthesizer is speaking. For example, if the user does not give a NEXT SENDER command soon enough, the command will be executed after the voice message starts.

The user-mail interface is designed for both experienced and inexperienced users. This is accomplished by defaulting all options in the program to a "normal" order. The "normal" order tells how many messages there are from the first sender, pauses, and then reads them all starting with a summary. This procedure is repeated for all of the senders until there are no more senders and then the program ends. This design lets an inexperienced user hear all his messages without knowing any commands, while the experienced user can issue commands for added flexibility.

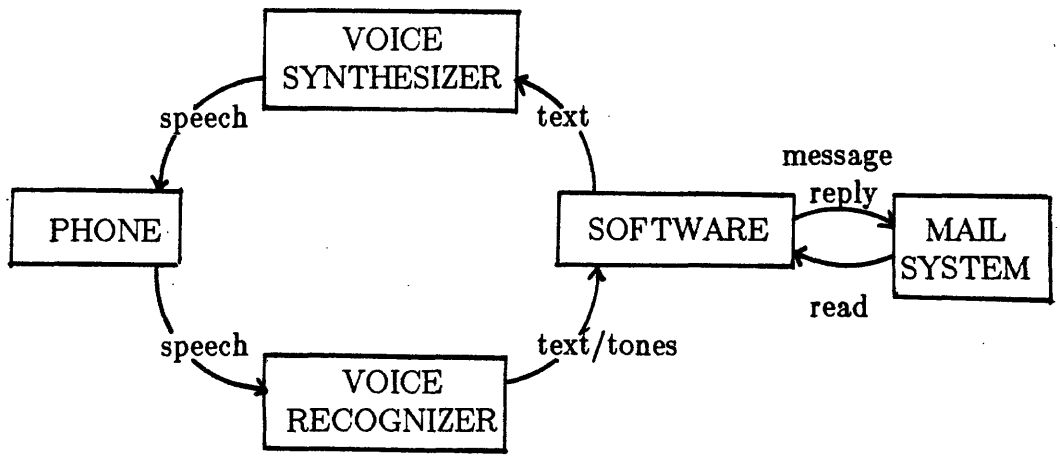
The number of possible commands is limited to twelve, which corresponds to the number of keys on a standard touch-tone telephone. This number was chosen because compound commands are difficult to remember. The commands allow the user to hear the next or previous message or sender, or to repeat messages. In addition, the user can pause in the middle of a message in order to increase comprehension.

Another feature of the interaction is pitch differentiation by the voice synthesizer. The synthesizer uses a low pitch when reading the message, and a higher pitch when reading control words, such as "Message 1". This helps the user distinguish between message text and control words.

Chapter 4
How Verbal Mail Works

To start the system, the user calls up the computer from a push-button telephone and enters his password. The Prose 2000 voice synthesizer [7] will prompt the user with a question, asking if he wants to hear his electronic mail or his phone messages. If the user selects his mail messages the verbal mail system is called.

The interface between the software for the verbal mail and the rest of the system is shown below.



The verbal mail system starts by asking the user if he wants to hear only his new messages. Then it states how many messages there are: "You have 15 messages." It then tells how many messages there are from the first sender and waits one second for a command: "5 from Caren <pause>." The

one second waiting period allows the user to alter the "normal" order (which will be described later). There are twelve commands, each initiated with a different button on the telephone (Figure 4-1) or the user's voice. Each command does the following:

BUTTON	ACTION
NEXT MESSAGE	Goes to next message of sender, if last message goes to next sender
BACKUP MESSAGE	Goes back to previous message (if at the start of message) Goes back to start of message (if in the middle of message)
REPEAT	Repeats the last phrase spoken
NEXT SENDER	Goes to the next sender
BACKUP SENDER	Goes back to beginning of current sender (if past the first message) Goes to previous sender (if on first message)
MORE INFO	Tells the full name and user name of the sender and the date and time of the message
ANS: YES	Replies yes to the sender's message
ANS: NO	Replies no to the sender's message
CALL ME AT	Replies to sender's message with the message to call back at a number
PAUSE	Pauses the program
CONTINUE	Continues in the "normal" order
QUIT	Exits program

If no response is given, the first sender's messages are read, the default order. The message starts with the prompt "message 1 <pause>" and waits one second for a command. If no command is given, the synthesizer reads the summary of a long message (more than 150 characters) or reads the entire text of a short message. The summary starts with either "it's about:," if the sender defined a subject, or "it begins:" if the summary was created from the

NEXT MESSAGE	BACKUP MESSAGE	REPEAT
NEXT SENDER	BACKUP SENDER	MORE INFO
ANS: YES	ANS: NO	CALL ME AT
PAUSE	CONTINUE	QUIT

Figure 4-1: Commands for telephone key pad

message. After the summary, there is another pause. The default is to read the message. After the message is read the next message of the same sender begins.

While hearing the message text the user can give any command, and the system will immediately respond. This option allows the user to stop in the middle of the message if he has heard enough, or pause temporarily to find out more information or write something down. If he wants to continue, the message will restart where it left off.

The user can have phases repeated by giving the REPEAT command. The first time something is repeated it is spoken slower. If the user still cannot understand, he can invoke REPEAT again, and the words will be spelled.

The user can get more information about the message by giving the MORE INFO command, which gives the full name and user name of the sender, and the date the message was sent. This information is not said each time, but is always available on request. The date format is determined by how long ago the message was sent. If it was sent more than a week ago, the month and date are returned without the time. If it was sent less than a week but before yesterday, the day and time is returned, for example, "Wednesday 2:15 p.m." If it was sent yesterday or today, "yesterday" or "today" and the time are returned.

The user can answer any message with a simple response using the responding keys: "Ans: Yes," "Ans: No," "Call Me At." The computer first asks if the user wants to send the message to make sure it understood him correctly. If the answer is yes (ANS: YES) a message with predefined text is

sent to the sender. The text begins: "I called in and read the message you sent <date of message> about <subject of message>." It then says one of the following: "The answer is yes," "The answer is no," or "Please call me at <number to be entered by user>."

When the user is done hearing all the messages from one sender, the next sender's messages automatically begins: "2 from Chris <pause>," and the process continues. If he wants, he can return to the first sender by giving the BACKUP SENDER command. If he just wants to repeat the last message he can use the BACKUP MESSAGE command.

After hearing all the senders the computer asks if the user want to continue: "There are no more senders. Do you want to repeat the senders? <pause>." The program ends unless a CONTINUE command is given.

A person can exit the program at anytime by giving the QUIT command.

A sample dialog is as follows:

Do you want to hear only your new messages? <pause> ANS: YES
You have 6 new messages <pause>
3 from Chris <pause>
Message 1 <pause>
It's about the Prose 2000 driver *<pause>*
<message text>
Message 2 <pause>
It's about Meeting *<pause>*
We will have a short meeting today at noon to discuss your progress. Is that time ok? ANS: YES
Do you want an affirmative reply sent? <pause> ANS: YES
Message was sent.
Message 3 <pause> NEXT SENDER
2 from Barry <pause>
Message 1 <pause> How's mail coming? Any new REPEAT

Hoooww'sss mmmaillll cccooommmingggg? REPEAT
H-o-w-s m-a-i-l c-o-m-i-n-g? Any new successes or problems?
Message 2 <pause> NEXT MESSAGE
1 from Caren <pause> Remember to MORE INFO
Caren H. Baker at MIT pamela, yesterday at 2 p.m. BACKUP SENDER
2 from Barry <pause> BACKUP SENDER
3 from Chris <pause>
Message 1 <pause> CALL ME AT
Do you want a return phone number reply sent? ANS: YES
Please enter a number after each tone ending with a number sign. Enter a star to cancel message
beep 7 *beep* 3 *beep* 7 *beep* 2 *beep* #
Is this number correct? 7372 ANS: YES
QUIT
Goodbye

Chapter 5
Performance Evaluation

The verbal mail system has a good interface and wastes as little time as possible. However, because of the hardware limitations, the response time is slower than desired. Some of the limitations were compensated for, while others still affect the system.

For novice users, the pronounciation of the text-to-speech voice synthesizer is difficult to understand. Also, because of the irrregular rules of pronounciation in the English language, the synthesizer often mispronounces words. For example, "message" is pronounced "mes-ig". A subroutine that contains a short dictionary of common words with their pronounciations was created to correct this problem. Another compensation for the pronounciation is the availability of the REPEAT command, which first slows down the speech and then spells the words. One advantage of the Prose 2000 over the other text-to-speech synthesizers is that it pronounces whole sentences using normal inflections rather than monotones [1,7].

A voice recognizer is used to recognize the touch-tones even though a integrated circuit could perform this task faster. Although the response time is slower, the voice recognizer was chosen because it allows for voice input. The user can issue commands by either hitting a button or speaking into the telephone. In this way, the user-mail interface is simplified.

The voice recognizer sometimes makes mistakes and must be trained for both the specific user's voice and for touch-tones. Training involves the user

reading each command several times into the voice recognizer. Also the recognizer does not always understand verbal commands in the presence of noise on the phone. If the spoken command is not understood, the user can reissue the command using the phone's key pad. The touch-tone sounds are easily distinguished even over the noisiest phone lines, so the tone command will almost always be understood.

In the future the user should be able to speak a message into the telephone and the computer will change it to text and mail it to the appropriate person [2,3]. It is advantageous to hear a message and immediately respond with any reply. Ease of replying is a quality that makes electronic mail superior to written mail.

Another improvement is to increase the dictionary size. This will improve the system, by making the synthesizer more intelligible.

References

[1] Allen, J. "Linguistic-based algorithms offer practical text-to-speech systems." Speech Technology. Fall 1981, v. 1, no. 1, pp. 12-16.

[2] Bahl, L. R., Cole, A.G., Jelinck, F., Mercer, R.L., Nadar, A., Nahamoo, Picheng, M.A. "Recognition of Isolated-Word Sentances from a 5000-Word Vocabulary Office Correspondence Task." Proc. 1983 IEEE Intl. Conf. on Acoust. Speech and Signal Proc. April 1983, Boston MA, pp. 1065-1067.

[3] Das, K. S. "Some Dimensionality Reduction Studies in Continuous Speech Recognition." Proc. 1983 IEEE Intl. Conf. on Acoust. Speech and Signal Proc. April 1983, Boston MA, pp. 292-295.

[4] Holden, J. B. "Experiences of and electronic mail vendor." National Computer Conference. May 1980, pp. 493-497.

[5] Hunt M. J. "Further Experiments in Text-Independent Speaker Recognition over Communication Channels." Proc. 1983 IEEE Intl. Conf. on Acoust. Speech and Signal Proc. April 1983, Boston MA, pp. 563-566.

[6] Pierrehumbert, J. "Synthesizing intonations." J. Acoust. Soc. Am. Oct 1981, v. 70, no. 4, pp. 985-995.

[7] Prose 2000 Text-to-Speech Converter User's Manual. Telesensory Systems, Inc. Palo Alto, CA, 1982.

[8] Saxton, W. A., Edwards, M. "Voice mail comes of age." Infosystems Aug 1980, v. 28, no. 8, p 72.

[9] Tomanek, G. "Implementing electronic mail in a telephone system: more that just talk." National Computer Conference. May 1980, pp. 527-531.

Appendix
Code Documentation

There are thirteen procedures that run the verbal mail system. The two start up procedures are Verbal_mail and Pointer. Six data procedures: Top_summ, Sort_senders, Sender, Get_mail_header, Sen_summ, and Dater that rearrange the mail messages into organized structures. Finally there are five procedures: Ask_sender, Ask_mess, Proc_mess, Get_tone, and Wait control the actions of the system.

The flow of the system is to initiate the pointers, obtain and organize the information, tell the messages, and then terminate the pointers.

The two start up procedures are:

Verbal_mail or vw

```
dcl verbal_mail entry;
call verbal_mail();
```

This is the driver of the system. It controls the basic flow of the system. It starts by asking for the user's name and the pasla name. It sets the username to the current user's with chuname. This is because when replies are sent the correct sender name must be placed in the from field. The username is restored at the end of the program. This procedure attaches the prose to the requested pasla. It calls pointer$init to initiate all the pointers. It then calls sender to get the information from the messages and top_summ to sort the messages. Then ask_sender, the procedure with the user interaction, is called. Pointer$term is called at the end to terminate the pointers.

Pointer:
Pointer$init

```
dcl pointer$init entry (char[168]vary, ptr, ptr, fix[31]);
call pointer$init (mbx_name, mbx_ptr, info_ptr, code);
```

This initiates all the pointers. It calls get_mail_header which

makes a copy of the user's mailbox and returns a pointer to the copy. It also finds a segment for the header_info structure and returns a pointer to this segment called info_ptr.

Pointer$term

```
dcl pointer$term entry (ptr);
call pointer$term (mbx_ptr);
```

Terminates the temporary mailbox and the segment where header_info is stored.

The mail messages are originally stored in the user's mailbox as one long string. The six data procedures extract information from each message and store it in structures. The data is organized using two structures: header_info and sort. Header_info contains the entire message organized so that each part is stored separately. Sort contains information about each sender. These two structures contain all the necessary information to recontruct the message. The control procedures use these structures to read the message, instead of the actual message stored in the user's mailbox.

The procedure sender takes the information from Get_mail_header, restructures it, and stores it in the header_info. The structure header_info is:

```
1 Header_info
   2 Message (1)
      3 Summary          char[168]vary   /* The summary used: either the
                                            subject or the start of the
                                            message */
      3 Sum_typ          char[12]vary    /* either "It begins" or
                                            "It's about" */
      3 Date_time                        /* of the form: */
         4 Long          char[64]vary    /* Monday 25 July 1983 */
         4 Day           char[16]vary    /* Monday */
         4 Date          char[16]vary    /* 25 */
         4 Month         char[16]vary    /* July */
         4 Year          char[16]vary    /* 1983 */
         4 Time          char[16]vary    /* 02:56:30 */
         4 Mdh_hms       char[20]vary    /* mm-dd-yy hh:mm:ss */
      3 From
         4 Long          char[64]vary    /* whole name and username */
         4 Name          char[32]vary    /* first name */
         4 User          char[64]vary    /* username */
      3 Re               char[168]vary   /* Subject field of message */
      3 Message
         4 Point         ptr             /* pointer to start of text */
         4 Len           fix             /* length of message */
      3 Seen             char[32]vary    /* "Viewed" or "Not Viewed" */
```

All the information about the message is stored in header_info. To save space the actual message text is not stored in the structure, only a pointer to the beginning and the length of it is stored. The Date and Sender are stored in different forms so they can be used alone later any of the ways. This structure is stored in its own segment mail_info that is pointed to by info_ptr.

The sort structure contains information about each sender and is used to sort the messages by sender, and the senders by most messages sent. The structure is:

```
1 Sort
    2 Total_mess_count fix              /* Total number of messages */
    2 Num_senders      fix              /* Total number of senders */
    2 Old_mess         fix              /* Number of viewed messages */
    2 Senders (25)
        3 Name         char[64]vary     /* First name of sender */
        3 Long_name    char[168]vary    /* Full name and user name */
        3 Mess_count   fix              /* Number of messages from the
                                           sender */
        3 Mess_num (25) fix             /* Actual message number of each
                                           message corresponding to
                                           number in header_info. */
    2 Current_sender   char[64]vary     /* Name of current sender */
    2 Long_sender      char[168]vary    /* Full name of current sender */
    2 Sender_num       fix              /* Index into sender array */
    2 Quit             bit(1)           /* Set to 1 if quit */
    2 Backup           bit(1)           /* Set to 1 if backing up */
    2 Priority (25)    fix              /* Array of the priority of each
                                           sender */
```

Associated with each sender is his name, the number of messages he sent, and an array of the actual message numbers in the mailbox of his messages. Also associated with each sender is a priority found in the priority array. The priority is determined by the number of messages a sender has sent. There is a one to one correspondence between senders and priority. The first sender is determined by sort.sender(sort.priority(1)). The actual message number of this sender's first message is found in sort.sender(sort.priority(1)).mess_number(1). Once this number is determined the information about the message can be obtained in header_info.msg(the number). These two structures contain all the necessary information about the messages.

The six procedures that organizes the data into the structures are:

Sender

```
dcl sender entry (ptr, fix, ptr, fix[31]);
call sender (mbx_ptr, entry_number, info_ptr, code);
```

Calls get_mail_header$get_nth, which creates a structure called header containing the pointers and lengths of each part of the message for one message. Using header, it creates a structure containing the text from different parts of the message's header, and a pointer and length of the message text. It stores this in the structure header_info. Sender calls sen_summ to create a summary for each message.

Sen_summ

```
dcl sen_summ entry (fix, ptr);
call sen_summ (entry_number, info_ptr);
```

Creates a summary of the message from the text, if one is not defined, and stores it in header_info.msg().summary. The summary is created from either the first six words, the first line, or the first sentance, whichever is shortest.

Dater

```
dcl dater entry (char[15]vary, char[20]vary, char[32]vary, char[32]vary);
call dater (weekday, mdy_hms, day_to_use, time);
```

Dater figures out the date and time of message compared to today. (i.e. Today, Yesterday, Tuesday...). It takes the weekday and date-time abbreviation and returns normal time and tells which date to use.

Top_summ

```
dcl top_summ entry (ptr, , bit(1), fix);
call top_summ (info_ptr, sort, new, count);
```

This procedure sets up the structure sort that contains information about each sender. It calls sort_senders to sort the senders, and then determines the priority of each sender. This priority is stored in sort.priority().

Sort_senders

```
dcl sort_senders entry ();
call sort_senders (sort);
```

Sort_sender sorts the messages by sender. It stores the name of each sender, the number of messages he sent, and the actual message number of each message in the sort structure.

After all of the data procedures have sorted the message, the contol procedures are called. The actions of the system is mainly controlled by ask_sender and ask_mess. These procedures keep track of the state of the system and perform the different commands. They determine what the voice synthesizer will say. They use the procedures Get_tone, Wait, and Proc_mess. The control procedures do the following:

Ask_sender

```
dcl ask_sender entry ( , ptr, bit(1), bit(1))
call ask_sender (sort, info_ptr, term, nec)
```

This procedures takes the two structures: sort and header_info (based info_ptr). It also takes two booleans: term and nec. Term is true if the terminal is being used, not the prose, and nec is true if the phone buttons and the nec is being used, not the terminal keys. These options are there to help testing the system.

Ask_sender controls which sender's messages are being heard. It starts with the sender with the highest priority (sort.priority(1)). The sender is changed with the commands. The procedure contains one large "do while" loop that keeps checking with "if" statements for different values of the variable "ans". The variable "ans" represents the command that the user has given. If there was no command given, that is ans = " ", then the current sender's messages are read. If the command is not understood, the sender does not change. When a QUIT command is given, the program ends, and returns to the top procedure. After the procedure has gone through all of the senders it asks if they should be repeated. To read the message, this procedure calls ask_mess. It passes the two structures sort and header_info, the two booleans, and the sender number to ask_mess.

Ask_mess

```
dcl ask_mess entry ( , ptr, fix, bit(1), bit(1))
```

call ask_mess (sort, info_ptr, sender_num, term, nec)

This procedure keeps track of which message you are hearing from one sender. Like in ask_sender there is one large "do while" loop that is continuously checking for the value of "ans". However each command does not always have the same affect. The effect of the command depends on what part the message is up to. There are four different states that the message can be in. They are:

State 0	At the start of the message
State 1	After message number is stated
State 2	After the summary is stated
State 3	At the end of the message

The variable "state" contains the current state. The system can also be in the middle or at the beginning of the message. There is a boolean msg.middle that keeps track of this. Some of the commands change the state of the system, while others do not. The state must be recorded in order to return to the right place. The commands NEXT MESSAGE, BACKUP MESSAGE, NEXT SENDER, BACKUP SENDER, and QUIT change the state of the system. The others cause an action and return to the process to the original place.

When the message is to be read the procedure proc_mess is called. It is passed a structure msg, that contains information about the message.

Proc_mess

dcl proc_mess entry (, char[1]vary, bit(1), bit(1))
call ask_mess (msg, ans, term, nec)

This procedure processes the message before it is sent out to the voice synthesizer. It splits the message up by the punctuation into phrases. There is an index sent after each phase to keep track of how much was said. If the message is interrupted, the index can be retrieved to determine where it is stopped. All of the information about the message is stored in a structure called msg. The phases are parsed before they are sent to the voice synthesizer for words that have to be corrected for pronounciation.

The structure msg is passed between ask_mess and proc_mess to help

process the the message. The stucture is as follows:

```
1 msg
    2 s_ptr              ptr      /* Pointer to start of message */
    2 len                fix      /* Length of message */
    2 proc_len           fix      /* Length of message processed
                                     (sent to prose with an index) */
    2 done               bit(1)   /* True if done with message */
    2 middle             bit(1)   /* True if in the middle of message */
    2 total_offset       fix      /* Number of indexes sent */
    2 offset (50)        fix      /* Offset from start of message */
    2 index_no (50)      fix      /* Index number associated with
                                     each offset */
    2 stop_index         fix      /* Index where stopped */
    2 stop_offset        fix      /* Associated offset from start of
                                     message where Stopped */
    2 repeat_offset      fix      /* Offset for repeating from
                                     when message stopped */
    2 repeat
      3 on               bit(1)   /* True if want to repeat */
      3 state            fix      /* The state that is repeated */
      3 num              fix      /* The message number repeated */
      3 len              fix      /* The length of message */
      3 times            fix      /* The times same thing is repeated */
      3 long             bit(1)   /* True if repeat 2 indexes,
                                     false if only one */
```

This structure contains information about the message. There are two arrays, offset and index_no, that contain information about each index. The offsets from the start of the message and the index number sent after the phrase are stored. There is also information telling if you are repeating or not. It records what was repeated and how many times it was.

Two procedures are used through out to get the input from the user.

```
Get_tone
     dcl get_tone entry (char[1]vary)
     call get_tone (ans)
```

This procedure gets a tone from the voice recognizer.

```
Proc_mess
      dcl wait entry (char[1]vary, bit(1))
      call wait (ans, nec)
```

This procedure waits one second, but checks for commands.